Pattern Recognition Letters 31 (2010) 91–99

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

On the memory complexity of the forward–backward algorithm

Wael Khreich [a,*], Eric Granger [a], Ali Miri [b], Robert Sabourin [a]

[a] Laboratoire d'imagerie, de vision et d'intelligence artificielle (LIVIA), École de technologie supérieure, Montreal, QC, Canada
[b] School of Information Technology and Engineering (SITE), University of Ottawa, Ottawa, ON, Canada

ARTICLE INFO

Article history:
Received 3 March 2009
Received in revised form 17 September 2009
Available online 24 September 2009

Communicated by O. Siohan

Keywords:
Hidden Markov Models
Forward–backward
Baum–Welch
Forward Filtering Backward Smoothing
Complexity analysis

ABSTRACT

The Forward–backward (FB) algorithm forms the basis for estimation of Hidden Markov Model (HMM) parameters using the Baum–Welch technique. It is however, known to be prohibitively costly when estimation is performed from long observation sequences. Several alternatives have been proposed in literature to reduce the memory complexity of FB at the expense of increased time complexity. In this paper, a novel variation of the FB algorithm – called the Efficient Forward Filtering Backward Smoothing (EFFBS) – is proposed to reduce the memory complexity without the computational overhead. Given an HMM with N states and an observation sequence of length T, both FB and EFFBS algorithms have the same time complexity, $\mathcal{O}(N^2T)$. Nevertheless, FB has a memory complexity of $\mathcal{O}(NT)$, while EFFBS has a memory complexity that is independent of T, $\mathcal{O}(N)$. EFFBS requires fewer resources than FB, yet provides the same results.

1. Introduction

The Hidden Markov Model (HMM) is a stochastic model for sequential data. Provided with an adequate number of states and sufficient set of data, HMMs are capable of representing probability distributions corresponding to complex real-world phenomena in term of simple and compact models. The forward–backward (FB) algorithm is a dynamic programming technique that forms the basis for estimation of HMM parameters. Given a finite sequence of training data, it efficiently evaluates the likelihood of this data given an HMM, and computes the smoothed conditional state probability densities – the sufficient statistics – for updating HMM parameters according to the Baum–Welch (BW) algorithm (Baum et al., 1970; Baum, 1972). BW is an iterative expectation maximization (EM) technique specialized for batch learning of HMM parameters via maximum likelihood estimation.

The FB algorithm is usually attributed to Baum et al. (1970), Baum (1972), although it was discovered earlier by Chang and Hancock (1966) and then re-discovered in different areas in the literature (Ephraim and Merhav, 2002). Despite suffering from numerical instability, the FB remains more famous than its numerically stable counterpart; the Forward Filtering Backward Smoothing (FFBS)[1] algorithm (Ott, 1967; Raviv, 1967). As a part of the BW algorithm, the FB algorithm has been employed in various applications such as signal processing, bioinformatics and computer security, (c.f., Cappe, 2001).

When learning from long sequences of observations the FB algorithm may be prohibitively costly in terms of time and memory complexity. This is the case, for example, of anomaly detection in computer security (Warrender et al., 1999; Lane, 2000), protein sequence alignment (Krogh et al., 1994) and gene-finding (Meyer and Durbin, 2004) in bioinformatics, and robot navigation systems (Koenig and Simmons, 1996). For a sequence of length T and an ergodic HMM with N states, the memory complexity of the FB algorithm grows linearly with T and N, $\mathcal{O}(NT)$, and its time complexity grows quadratically with N and linearly with T, $\mathcal{O}(N^2T)$.[2] For a detailed analysis of complexity the reader is referred to Table 1 (Section 3). When T is very large, the memory complexity may exceed the resources available for the training process, causing overflow from internal system memory to disk storage.

Several alternatives have been proposed in literature trying to reduce the memory complexity of the FB algorithm or, ideally, to eliminate its dependence on the sequence length T. Although there is an overlap between these approaches and some have been re-discovered, due to the wide range of HMMs applications, they can be divided into two main types. The first type performs exact computations of the state densities using fixed-interval smoothing (similar to FB and FFBS), such as the checkpointing and forward-only algorithms. However, as detailed in Section 3,

* Corresponding author.
E-mail addresses: wael.khreich@livia.etsmtl.ca (W. Khreich), eric.granger@etsmtl.ca (E. Granger), samiri@site.uottawa.ca (A. Miri), robert.sabourin@etsmtl.ca (R. Sabourin).

[1] FFBS is also referred to in literature as $\alpha - \gamma$ and disturbance smoothing algorithm.

[2] If the HMM is not fully connected, the time complexity reduces to $\mathcal{O}(NQ_{max}T)$, where Q_{max} is the maximum number of states to which any state is connected.

0167-8655/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2009.09.023

the memory complexity of the former algorithm still depends on T, while the latter eliminates this dependency at the expenses of a significant increase in time complexity, $\mathcal{O}(N^4T)$.

Although outside the scope of this paper, there are approximations to the FB algorithm. Such techniques include approximating the backward variables of the FB algorithm using a sliding time-window on the training sequence (Koenig and Simmons, 1996), or slicing the sequence into several shorter ones that are assumed independent, and then applying the FB on each sub-sequence (Yeung and Ding, 2003; Wang et al., 2004). Other solutions involve performing on-line learning techniques on a finite data however (LeGland and Mevel, 1997; Florez-Larrahondo, 2005). These techniques are not explored because they provide approximations to the smoothed state densities, and hence lead to different HMMs. In addition, their theoretical convergence properties are based on infinite data sequence ($T \to \infty$).

A novel modification to the FFBS called Efficient Forward Filtering Backward Smoothing (EFFBS) is proposed, such that its memory complexity is independent of T, without incurring a considerable computational overhead. In contrast with the FB, it employs the inverse of HMM transition matrix to compute the smoothed state densities without any approximations. A detailed complexity analysis indicates that the EFFBS is more efficient than existing approaches when learning of HMM parameters form long sequences of training data.

This paper is organized as follows. In Section 2, the estimation of HMM parameters using the FB and the Baum–Welch algorithms is presented. Section 3 reviews techniques in literature that addressed the problem of reducing the memory complexity of the FB algorithm, and accesses their impact on the time complexity. The new EFFBS algorithm is presented in Section 4, along with a complexity analysis and case study.

2. Estimation of HMM parameters

The Hidden Markov Model (HMM) is a stochastic model for sequential data. A discrete-time finite-state HMM is a stochastic process determined by the two interrelated mechanisms – a latent Markov chain having a finite number of states, and a set of observation probability distributions, each one associated with a state. At each discrete time instant, the process is assumed to be in a state, and an observation is generated by the probability distribution corresponding to the current state. The model is termed *discrete* (or finite-alphabet) if the output alphabet is finite. It is termed *continuous* (or general) if the output alphabet is not necessarily finite e.g., the state is governed by a parametric density function, such as Gaussian, Poisson, etc. For further details regarding HMM the reader is referred to the extensive literature (Ephraim and Merhav, 2002; Rabiner, 1989).

Formally, a discrete-time finite-state HMM consists of N hidden states in the finite-state space $S = \{S_1, S_2, \ldots, S_N\}$ of the Markov process. Starting from an initial state S_i, determined by the initial state probability distribution π_i, at each discrete-time instant, the process transits from state S_i to state S_j according to the transition probability distribution $a_{ij} (1 \leqslant i,j \leqslant N)$. As illustrated in Fig. 1, the process then emits a symbol v according to the output probability distribution $b_j(v)$, which may be discrete or continuous, of the current state S_j. The model is therefore parametrized by the set $\lambda = (\pi, A, B)$, where vector $\pi = \{\pi_i\}$ is initial state probability distribution, matrix $A = \{a_{ij}\}$ denotes the state transition probability distribution, and matrix $B = \{b_j(k)\}$ is the state output probability distribution.

2.1. Baum–Welch algorithm

The Baum–Welch (BW) algorithm (Baum and Petrie, 1966; Baum et al., 1970) is an Expectation–Maximization (EM) algorithm (Dempster et al., 1977) specialized for estimating HMM parameters. It is an iterative algorithm that adjusts HMM parameters to best fit the observed data, $o_{1:T} = \{o_1, o_2, \ldots, o_T\}$. This is typically achieved by maximizing the log-likelihood, $\ell_T(\lambda) \triangleq \log P(o_{1:T}|\lambda)$ of the training data over HMM parameters space (Λ):

$$\lambda^* = \operatorname{argmax}_{\lambda \in \Lambda} \ell_T(\lambda). \tag{1}$$

The subscripts $(\cdot)_{t|T}$ in formulas (2)–(4) are used to stress the fact that these are *smoothed* probability estimates. That is, computed from the whole sequence of observations during batch learning.

During each iteration, the E-step computes the sufficient statistics, i.e., the *smoothed a posteriori* conditional state density:

$$\gamma_{t|T}(i) \triangleq P(q_t = i|o_{1:T}, \lambda) \tag{2}$$

and the*smoothed a posteriori* conditional joint state density[3]:

$$\xi_{t|T}(i,j) \triangleq P(q_t = i, q_{t+1} = j|o_{1:T}, \lambda) \tag{3}$$

which are then used, in the M-step, to re-estimate the model parameters:

$$\begin{aligned} \pi_i^{(k+1)} &= \gamma_{1|T}^{(k)}(i) \\ a_{ij}^{(k+1)} &= \frac{\sum_{t=1}^{T-1} \xi_{t|T}^{(k)}(i,j)}{\sum_{t=1}^{T-1} \gamma_{t|T}^{(k)}(i)} \\ b_j^{(k+1)}(m) &= \frac{\sum_{t=1}^{T} \gamma_{t|T}^{(k)}(j) \delta_{o_t v_m}}{\sum_{t=1}^{T} \gamma_{t|T}^{(k)}(j)} \end{aligned} \tag{4}$$

The Kronecker delta δ_{ij} is equal to one if $i = j$ and zero otherwise.

Starting with an initial guess of the HMM parameters, λ^0, each iteration k, of the E- and M-step is guaranteed to increase the likelihood of the observations giving the new model until a convergence to a stationary point of the likelihood is reached (Baum et al., 1970; Baum, 1972).

The objective of the E-step is therefore, to compute an estimate $\hat{q}_{t|\tau}$ of an HMM's hidden state at any time t given the observation history $o_{1:\tau}$. The optimal estimate $\hat{q}_{t|\tau}$ in the minimum mean square error (MMSE) sense of the state q_t of the HMM at any time t, given the observation is the conditional expectation of the state q_t given $o_{1:\tau}$ (Li and Evans, 1992; Shue et al., 1998):

$$\hat{q}_{t|\tau} = E(q_t|o_{1:\tau}) = \sum_{i=1}^{N} \gamma_{t|\tau}(i). \tag{5}$$

In estimation theory, this conditional estimation problem is called *filtering* if $t = \tau$; *prediction* if $t > \tau$, and *smoothing* if $t < \tau$. Fig. 2 provides an illustration of these estimation problems. The smoothing problem is termed *fixed-lag* smoothing when computing the $E(q_t|o_{1:t+\Delta})$ for a fixed-lag $\Delta > 0$, and *fixed-interval* smoothing when computing the $E(q_t|o_{1:T})$ for all $t = 1, 2, \ldots, T$.

For batch learning, the state estimate is typically performed using fixed-interval smoothing algorithms (as described in Sections 2.2 and 2.3). Since it incorporates more evidence, fixed-interval smoothing provides better estimate of the states than filtering. This is because the latter relies only on the information that is available at the time.

2.2. Forward–backward (FB)

The forward–backward (Chang and Hancock, 1966; Baum et al., 1970; Baum, 1972) is the most commonly used algorithm for computing the smoothed state densities of Eqs. (2) and (3). Its forward

[3] To facilitate reading, the terms a posteriori and conditional will be omitted whenever it is clear from the context.

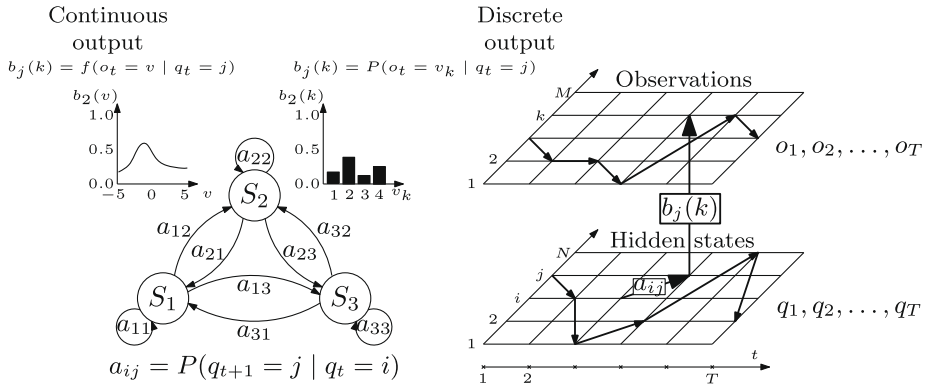

Fig. 1. Illustration of a fully connected (ergodic) three states HMM with either a continuous or discrete output observations (left). Illustration of a discrete HMM with N states and M symbols switching between the hidden states q_t and generating the observations o_t (right). $q_t \in S$ denotes the state of the process at time t with $q_t = i$ means that the state at time t is S_i.

pass computes the *joint* probability of the state at time t *and* the observations up to time t:

$$\alpha_t(i) \triangleq P(q_t = i, o_{1:t}|\lambda), \tag{6}$$

using the following recursion, for $t = 1, 2, \ldots, T$ and $j = 1, \ldots, N$:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(o_{t+1}). \tag{7}$$

Similarly, the backward path computes the probability of the observations from time $t+1$ up to T *given* the state at time t,

$$\beta_t(i) \triangleq P(o_{t+1:T}|q_t = i, \lambda), \tag{8}$$

according to the reversed recursion, for $t = T-1, \ldots, 1$ and $i = 1, \ldots, N$:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j). \tag{9}$$

Then, both smoothed state and joint state densities of Eqs. (2) and (3) – the sufficient statistics for HMM parameters update – are directly obtained:

$$\gamma_{t|T}(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{k=1}^{N} \alpha_t(k)\beta_t(k)}, \tag{10}$$

$$\xi_{t|T}(i,j) = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{k=1}^{N} \sum_{l=1}^{N} \alpha_t(k) a_{kl} b_l(o_{t+1}) \beta_{t+1}(l)}. \tag{11}$$

By definition, the elements of α are not probability measures unless normalized, that is why they are usually called the α-variables. Due to this issue, the FB algorithm is not stable and

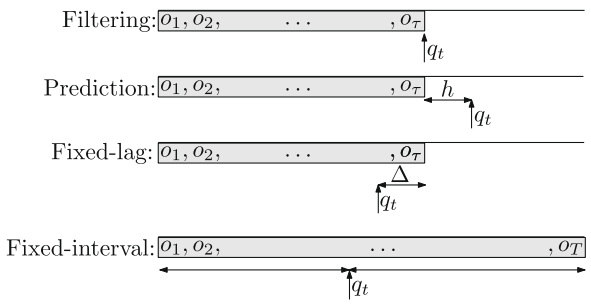

Fig. 2. Illustration of the filtering, prediction and smoothing estimation problems. Shaded boxes represent the observation history (already observed), while the arrows represent the time at which we would like to compute the state estimates.

susceptible to underflow when applied to a long sequence of observation. In (Levinson et al., 1983) and (Rabiner, 1989) the authors suggest scaling the α-variables by their summation at each time:

$$\bar{\alpha}_t(i) = \frac{\alpha_t(i)}{\sum_i \alpha_t(i)} = \frac{P(q_t = i, o_{1:t}\lambda)}{P(o_{1:t}\lambda)} = P(q_t = i|o_{1:t}, \lambda) \tag{12}$$

making for a *filtered* state estimate. This of course requires scaling the beta β values, which have no an intuitive probabilistic meaning, with the same quantities (refer to Algorithms 2 and 3 in Appendix A). Other solutions involves working with extended exponential and extended logarithm functions (Mann, 2006).

2.3. Forward Filtering Backward Smoothing (FFBS)

As highlighted in literature (Devijver, 1985; Ephraim and Merhav, 2002; Cappe and Moulines, 2005), the FFBS is a lesser known but numerically *stable* alternative algorithm to FB, which has been proposed in (Ott, 1967; Raviv, 1967; Lindgren, 1978; Askar and Derin, 1981). In addition, the FFBS is probabilistically more meaningful since it propagates probability densities in its forward and backward passes.

During the forward pass, the algorithm splits the state density estimates into *predictive* (13) and *filtered* (14) state densities:

$$\gamma_{t|t-1}(i) \triangleq P(q_t = i|o_{1:t-1}, \lambda), \tag{13}$$

$$\gamma_{t|t}(i) \triangleq P(q_t = i|o_{1:t}, \lambda). \tag{14}$$

The initial predictive state density is the initial probability distribution, $\gamma_{1|0}(i) = \pi_i$. For $t = 1, \ldots, T$, the filtered state density at time t is computed from the predictive state density at time $t-1$,

$$\gamma_{t|t}(i) = \frac{\gamma_{t|t-1}(i) b_i(o_t)}{\sum_{j=1}^{N} \gamma_{t|t-1}(j) b_j(o_t)} \tag{15}$$

and then, the predictive state density at time $t+1$ is computed from the filtered state density at time t:

$$\gamma_{t+1|t}(j) = \sum_{i=1}^{N} \gamma_{t|t}(i) a_{ij}. \tag{16}$$

Both the filtered and one-step prediction density, follow from a combination of the Bayes' rule of conditional probability and the Markov property.

Backward recursion, leads directly to the required *smoothed* densities by solely using the filtered and predictive state densities (Lindgren, 1978; Askar and Derin, 1981) for $t = T-1, \ldots, 1$:

Table 1
Worst-case time and memory complexity analysis for the FB and FFBS for processing an observation sequence $o_{1:T}$ of length T with an N state HMM. The scaling procedure as described in (Rabiner, 1989) and shown in Appendix A (Algorithms 2 and 3) is taken into consideration with the FB algorithm.

Computations	Time			Memory
	# Multiplications	# Divisions	# Additions	
Forward–backward (FB)				
α_t	$N^2T + NT - N^2$	$NT - N$	$N^2T - N^2 + N$	NT
β_t	$2N^2T - 2N^2$	NT	$N^2T - NT - N^2 + N$	–
$\sum_t^T \gamma_{t\|T}$	NT	NT	$NT - T$	N
$\sum_{t=1}^{T-1} \xi_{t\|T}$	$3N^2T - 3N^2$	$N^2T - N^2$	$N^2T - NT - N^2 + N$	N^2
Total	$6N^2T + 2NT - 6N^2$	$N^2T + 3NT - N^2 - N$	$3N^2T + NT - 3N^2 + 3N - T$	$NT + N^2 + N$
Forward Filtering Backward Smoothing (FFBS)				
$\gamma_{t\|t}$	$N^2T + NT - N^2$	NT	$N^2T - N^2 + N$	NT
β_t	–	–	$N^2T - NT - N^2 + N$	–
$\sum_t^T \gamma_{t\|T}$	–	–	$NT - T$	N
$\sum_{t=1}^{T-1} \xi_{t\|T}$	$2N^2T - 2N^2$	$N^2T - N^2$	$N^2T - NT - N^2 + N$	N^2
Total	$3N^2T + NT - 3N^2$	$N^2T + NT - N^2$	$3N^2T + NT - 3N^2 + 3N - T$	$NT + N^2 + N$

$$\xi_{t|T}(i,j) = \frac{\gamma_{t|t}(i)a_{ij}}{\sum_{i=1}^{N}\gamma_{t|t}(i)a_{ij}}\gamma_{t+1|T}(j), \tag{17}$$

$$\gamma_{t|T}(i) = \sum_{j=1}^{N}\xi_{t|T}(i,j). \tag{18}$$

In contrast with the FB algorithm, the backward pass of the FFBS does *not* require access to the observations. In addition, since it is propagating probabilities densities, in both forward and backward passes, the algorithm is numerically stable. In the forward pass, the *filtered* state density can be also computed in the same way as the normalized $\bar{\alpha}$-variables (12), since after the normalization $\bar{\alpha}$ becomes the state filter. When the forward pass is completed at time T, $\gamma_{T|T}$ is the only *smoothed* state density, while all previous ones are filtered and predictive state estimates. The reader is referred to Algorithms 4 and 5 in Appendix A for further details on the FFBS.

3. Complexity of Fixed-Interval Smoothing Algorithms

This section presents a detailed analysis of the time and memory complexity for the FB and FFBS algorithms. It also presents the complexity of the checkpointing and forward-only algorithms. These algorithms have been proposed to reduce the memory complexity of FB at the expense of time complexity.

The time complexity is defined as the sum of the worst-case running time for each operation (e.g., multiplication, division and addition) required to process an input. The growth rate is then obtained by making the parameters of the worst-case complexity tend to ∞. Memory complexity is estimated as the number of 32 bit registers needed during learning process to store variables. Only the worst-case memory space required during processing phase is considered.

3.1. FB and FFBS algorithms

The main bottleneck for both FB and FFBS algorithms occurs during the induction phase. In the ergodic case, the time complexity per iteration is $\mathcal{O}(N^2T)$ as shown in (7) for the FB algorithm (see also lines 6–11 of Algorithm 2 in Appendix A), and in (15) and (16) for the FFBS algorithm (see also lines 3–7 Algorithm 4 in Appendix A).

In addition, as presented in Fig. 3, the filtered state densities (Eqs. (7) for FB, and (15) and (16) for FFBS) computed at each time step of the forward pass, must be loaded into the memory in order to compute smoothed state densities (Eqs. (2) and (3)) in the backward pass. The bottleneck in terms of memory storage, for both algorithms, is therefore, the size of the matrix of $N \times T$ floating-points values.

A memory complexity of $\mathcal{O}(N^2T)$ is often attributed to FB and FFBS algorithms in the literature. In such analysis, memory is assigned to $\xi_{t|T}(i,j)$ which requires T matrices of $N \times N$ floating points. However, as shown in the transition update Eq. (4), only the summation over time is required; hence only one matrix of N^2 floating points is required for storing $\sum_{t=1}^{T}\xi_{t|T}(i,j)$. In addition, the N floating point values required for storing the filtered state estimate, $\gamma_{t|t}$, (i.e., $N^2 + N$), are also unavoidable.

Table 1 details an analysis of the worst-case time and memory complexity for the FB and FFBS algorithms processing an observation sequence $o_{1:T}$ of length T. Time complexity represents the worst-case number of operations required for one iteration of BW. A BW iteration involves computing one forward and one backward pass with $o_{1:T}$. Memory complexity is the worst-case number of 32 bit words needed to store the required temporary variables in RAM. Analysis shows that the FFBS algorithm requires slightly fewer computations ($3N^2T + NT$ multiplications, $2NT$ divisions and $3NT$ additions) than the FB algorithm. It can be also seen that both algorithms require the same amount of storage: an array of $NT + N^2 + N$ floating point values.

With both FB and FFBS algorithms a memory problem stems from its linear scaling with the length of the observation sequence T. Applications that require estimating HMM parameters from a long sequence will require long training times and large amounts of memory, which may be prohibitive for the training process.

As described next, some alternatives for computation of exact smoothed density have been proposed to alleviate this issue. However, these alternatives are either still partially dependent on the sequence length T, or increase the computational time by orders of magnitude to achieve a constant memory complexity.

3.2. Checkpointing algorithm

The checkpointing algorithm stores only some reference columns or checkpoints, instead of storing the whole matrix of filtered state densities (Grice et al., 1997; Tarnas and Hughey, 1998). Therefore, it divides the input sequence into $\sqrt{T}$ sub-sequences. While performing the forward pass, it only stores the first column of the forward variables for each sub-sequences. Then, during the

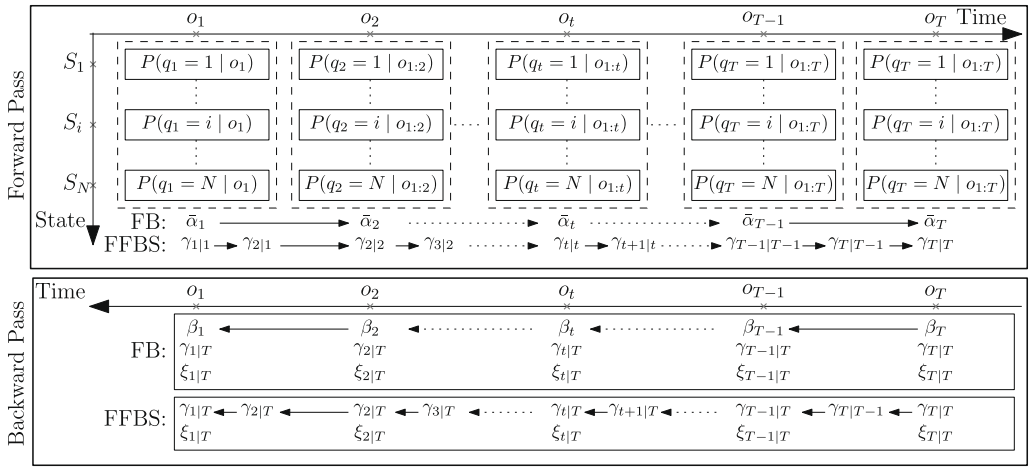

Fig. 3. An illustration of the values that require storage during the forward and backward passes of the FB and FFBS algorithms. During the forward pass the filtered state densities at each time step ($N \times T$ floating-points values) must be stored into the internal system memory, to be then used to compute the smoothed state densities during the backward pass.

backward pass, the forward values for each sub-sequence are sequentially recomputed, beginning with their corresponding checkpoints. Its implementation reduces the memory complexity to $\mathcal{O}(N\sqrt{T})$, yet increases the time complexity to $\mathcal{O}(N^2(2T-\sqrt{T}))$. The memory requirement can be therefore traded-off against the time complexity.

3.3. Forward-only algorithm

A direct propagation of the smoothed densities in forward-only manner is an alternative that has been proposed in the communication field (Narciso, 1993; Sivaprakasam and Shanmugan, 1995; Elliott et al., 1995) and re-discovered in bioinformatics (Miklos and Meyer, 2005; Churbanov and Winters-Hilt, 2008). The basic idea is to directly propagate all *smoothed* information in the forward pass

$$\sigma_t(i,j,k) \triangleq \sum_{\tau=1}^{t-1} P(q_\tau = i, q_{\tau+1} = j, q_t = k | o_{1:t}, \lambda), \tag{19}$$

which represents the probability of having made a transition from state S_i to state S_j at some point in the past ($\tau < t$) and of ending up in state S_k at the current time t. At the next time step, $\sigma_{t+1}(i,j,k)$ can be recursively computed from $\sigma_t(i,j,k)$ using:

$$\sigma_t(i,j,k) = \sum_{n=1}^{N} \sigma_{t-1}(i,j,n) a_{nk} b_k(o_t) + \alpha_{t-1}(i) a_{ik} b_k(o_t) \delta_{jk} \tag{20}$$

from which the smoothed state densities can be obtained, at time T, by marginalizing over the states.

By eliminating the backward pass, the memory complexity becomes $\mathcal{O}(N^2+N)$, which is independent T. However, it is achieved at the expenses of increasing the time complexity to $\mathcal{O}(N^4T)$ as can be seen in the four-dimensional recursion (20). Due to this computational overhead for training, this forward-only approach has not gained much popularity in practical applications.

4. Efficient Forward Filtering Backward Smoothing (EFFBS) algorithm

In this section, a novel and efficient alternative that is able to compute the exact smoothed densities with a memory complexity that is independent of T, and without increasing time complexity is described. The proposed alternative – termed the Efficient Forward Filtering Backward Smoothing (EFFBS) – is an extension of the FFBS algorithm. It exploits the facts that a backward pass of the FFBS algorithm does not require any access to the observations and that the smoothed state densities are recursively computed from each other in a backward pass.

Instead of storing all the predictive and filtered state densities (see Eqs. (13) and (14)) for each time step $t = 1, \ldots, T$, the EFFBS recursively recomputes their values starting from the end of the sequence. Accordingly, the memory complexity of the algorithm becomes independent of the sequence length. That is, by storing only the last values of $\gamma_{T|T}$ and $\gamma_{T|T-1}$ from the forward pass, the previously-computed filtered and predictive densities can be recomputed recursively, in parallel with the smoothed densities (refer to Algorithm 1).

Let the notations $\odot$ and $(\div)$ be the term-by-term multiplication and division of two vectors, respectively. Eqs. (14) and (16) can be written as:

$$\gamma_{t|t} = \frac{\left[\gamma_{t|t-1} \odot b_t\right]}{\left\|\gamma_{t|t}\right\|_1}, \tag{21}$$

$$\gamma_{t|t-1} = A' \gamma_{t-1|t-1}, \tag{22}$$

where $b_t = [b_1(o_t), \ldots b_N(o_t)]'$, $\|\cdot\|_1$ is the usual 1-norm of a vector (column sum) and the *prime* superscript denotes matrix transpose. The backward pass of the EFFBS algorithm starts by using only the column-vector filtered state estimate of the last observation, $\gamma_{T|T}$, resulting from the forward pass. Then, for $t = T-1, \ldots 1$, $\gamma_{t|t-1}$ is recomputed from $\gamma_{t|t}$ using:

$$\gamma_{t|t-1} = \frac{\left[\gamma_{t|t}(\div) b_t\right]}{\left\|\gamma_{t|t-1}\right\|_1} \tag{23}$$

from which, the filtered state estimate can be recomputed by

$$\gamma_{t-1|t-1} = \left[A'\right]^{-1} \gamma_{t|t-1}. \tag{24}$$

This step requires the inverse of the transposed transition matrix $[A']^{-1}$ to solve N linear equations with N unknowns. The only necessary condition is the nonsingularity of the transition matrix A to be invertible. This is a weak assumption that has been used in most of

Algorithm 1: EFFBS($o_{1:T}, \lambda$): Efficient Forward Filtering Backward Smoothing

Output: $\gamma_{t|T}(i) = P(q_t = i \mid o_{1:T})$ and $\xi_{t|T}(i,j) = P(q_t = i, q_{t+1} = j \mid o_{1:T})$

/* Forward Filtering: the filtered, $\gamma_{t|t}$, and predictive, $\gamma_{t+1|t}$, state densities are column vectors of length N. Their values are recursively computed, however they are not stored at each time step, t, but recomputed in the backward pass to reduce the memory complexity. */

for $i = 1, \ldots, N$ **do** /* Initialization */

  $\gamma_{1|0}(i) = \pi_i$

for $t = 1, \ldots T-1$ **do** /* Induction */

  for $i = 1, \ldots, N$ **do**

    $\gamma_{t|t}(i) \leftarrow \frac{\gamma_{t|t-1}(i) b_i(o_t)}{\sum_{k=1}^{N} \gamma_{t|t-1}(k) b_k(o_t)}$

  for $j = 1, \ldots, N$ **do**

    $\gamma_{t+1|t}(j) \leftarrow \sum_{i=1}^{N} \gamma_{t|t}(i) a_{ij}$

for $t = 1, \ldots T$ **do** /* Evaluation */

  $\ell_T(\lambda) \leftarrow \ell_T(\lambda) + \log \sum_{i=1}^{N} b_i(o_t) \gamma_{t|t-1}(i)$

/* Backward Smoothing: the last values of the filtered, $\gamma_{T|T}$, and predictive, $\gamma_{T|T-1}$, state densities are now used to recompute their previous values. */

$A = a^{-1}$ /* matrix inversion only once per iteration */

for $t = T-1, \ldots, 1$ **do** /* Induction */

  for $i = 1, \ldots, N$ **do**

    $\gamma_{t|t}(i) = \sum_{k=1}^{N} A_{ki} \gamma_{t+1|t}(k)$

    $\gamma_{t|t-1}(i) = \frac{\gamma_{t|t}(i) / b_i(o_t)}{\sum_{k=1}^{N} \gamma_{t|t}(k) / b_k(o_t)}$

    for $j = 1, \ldots, N$ **do**

      $\xi_{t|T}(i,j) \leftarrow \frac{\gamma_{t|t}(i) a_{ij}}{\sum_{i=1}^{N} \gamma_{t|t}(i) a_{ij}} \gamma_{t+1|T}(j)$

    $\gamma_{t|T}(i) \leftarrow \sum_{j=1}^{N} \xi_{t|T}(i,j)$

the studies that addressed the statistical properties of HMMs (c.f. (Ephraim and Merhav, 2002)). In particular, it holds true in applications that use fully connected HMMs, where the elements of the transition matrix are positives (i.e., irreducible and aperiodic).

4.1. Complexity analysis of the EFFBS algorithm

In order to compute the smoothed densities in a constant memory complexity, the EFFBS algorithm must recompute $\gamma_{t|t}$ and $\gamma_{t|t-1}$ values in its backward pass, instead of storing them, which requires twice the effort of the forward pass computation. This recomputation however, takes about the same amount of time required for computing the β values in the backward pass of the FB algorithm.

Table 2 presents a worst-case analysis of time and memory complexity for the EFFBS. The proposed algorithm is comparable to both FB and FFBS algorithms (see Table 1) in terms of computational time, with an overhead of about $N^2T + NT$ multiplications and additions. An additional computational time is required for the inversion of the transition matrix, *which is computed only once for each iteration.* The

Table 2
Worst-case time and memory complexity of the EFFBS algorithm for processing an observation sequence $o_{1:T}$ of length T with an N state ergodic HMM.

Computation	Time			Memory
	# Multiplications	# Divisions	# Additions	
$\gamma_{T\|T}$	$N^2T + NT - N^2$	NT	$N^2T - N^2 + N$	N
$\gamma_{t\|t}$ and $\gamma_{t\|t-1}$	$N^2T + NT - N^2$	NT	$N^2T - N^2 + N$	$2N$
A^{-1}	$\frac{N^3}{3} + \frac{N^2}{2} - \frac{N}{3}$	–	$\frac{N^3}{3} + \frac{N^2}{2} - \frac{5N}{6}$	N^2
$\sum_t^T \gamma_{t\|T}$	–	–	$N^2T - NT - N^2 + N$	N
$\sum_{t=1}^{T-1} \xi_{t\|T}$	$2N^2T - 2N^2$	$N^2T - N^2$	$N^2T - NT - N^2 + N$	N^2
Total	$4N^2T + 2NT + \frac{N^3}{3} - \frac{7N^2}{2} - \frac{N}{2}$	$N^2T + 2NT - N^2$	$4N^2T - 2NT + \frac{N^3}{3} - \frac{7N^2}{2} + \frac{19N}{6}$	$N^2 + 5N$

Table 3
Time and memory complexity of the EFFBS versus other approaches (processing an observation sequence $o_{1:T}$ of length T, for an ergodic HMM with N states).

Algorithm	Time	Memory
FB	$\mathcal{O}(N^2T)$	$\mathcal{O}(NT)$
FFBS	$\mathcal{O}(N^2T)$	$\mathcal{O}(NT)$
Checkpointing	$\mathcal{O}(N^2(2T-\sqrt{T}))$	$\mathcal{O}(N\sqrt{T})$
Forward-Only	$\mathcal{O}(N^4T)$	$\mathcal{O}(N)$
EFFBS	$\mathcal{O}(N^2T)$	$\mathcal{O}(N)$

classical Gauss–Jordan elimination, may be used for instance to achieve this task in $\mathcal{O}(N^3)$. Faster algorithms for matrix inversion could be also used. For instance, inversion based on Coppersmith–Winograd algorithm (Coppersmith and Winograd, 1990) is $\mathcal{O}(N^{2.376})$.

The EFFBS only requires two column vectors of N floating point values on top of the unavoidable memory requirements (N^2+N) of the smoothed densities. In addition, a matrix of N^2 floating point values is required for storing the inverse of the transposed transition matrix $[A']^{-1}$.

Table 3 compares the time and memory complexities of existing fixed-interval smoothing algorithms for computing the state densities to the proposed EFFBS algorithm.

4.2. Case study in host-based anomaly detection

Intrusion Detection Systems (IDSs) are used to identify, assess, and report unauthorized computer or network activities. Host-based IDSs (HIDSs) are designed to monitor the host system activities and state, while network-based IDSs monitor network traffic for multiple hosts. In either case, IDSs have been designed to perform misuse detection – looking for events that match patterns corresponding to known attacks – and anomaly detection – detecting significant deviations from normal system behavior.

Operating system events are usually monitored in HIDSs for anomaly detection. Since system calls are the gateway between user and kernel mode, early host-based anomaly detection systems monitor deviation in system call sequences. Various detection techniques have been proposed to learn the normal process behavior through system call sequences (Warrender et al., 1999). Among these, techniques based on discrete Hidden Markov Models (HMMs) have been shown to provide high level of performance (Warrender et al., 1999).

The primary advantage of anomaly-based IDS is the ability to detect novel attacks for which the signatures have not yet been extracted. However, anomaly detectors will typically generate false alarms mainly due to incomplete data for training. A crucial step to design an anomaly-based IDS is to acquire sufficient amount of data for training. Therefore, a large stream of system call data is usually collected from a process under normal operation in a secured environment. This data is then provided for training the parameters of an ergodic HMM to fit the normal behavior of the monitored process.

A well trained HMM should be able to capture the underlying structure of the monitored application using the temporal order of system calls generated by the process. Once trained, a normal sequence presented to HMM should produce a higher likelihood value than for a sequence that does not belong to the normal process pattern or language. According to a decision threshold the HMM-based Anomaly Detection System (ADS) is able to discriminate between the normal and anomalous system call sequences.

The sufficient amount of training data depends on the complexity of the process and is very difficult to be quantified *a priori*. Therefore, the larger the sequence of system calls provided for training, the better the discrimination capabilities of the HMM-based ADS and the fewer the rate of false alarms. In practice however, the memory complexity of the FB (or FFBS) algorithm could be be prohibitively costly when learning is performed for a large sequence of system calls, as previously described. In their experimental work on training HMM for anomaly detection using large sequences of system calls, Warrender et al. (1999) state:

> On our largest data set, HMM training took approximately two months, while the other methods took a few hours each. For all but the smallest data sets, HMM training times were measured in days, as compared to minutes for the other methods.

This is because the required memory complexity for training the HMM with the FB algorithm exceeds the resources available, causing overflows from internal system memory, e.g., Random-Access Memory (RAM), to disk storage. Accessing paged memory data on a typical disk drive is approximately one million time slower than accessing data in the RAM.

For example, assume that an ergodic HMM with $N=50$ states is trained using an observation sequence of length $T=100\times10^6$ system calls collected from a complex process. The time complexity per iteration of the FFBS algorithm (see Table 1):

$$Time_{\{FFBS\}} = 75.5\times10^{10}\,\mathrm{MUL} + 25.5\times10^{10}\mathrm{DIV} + 75.5\times10^{10}\mathrm{ADD}$$

is comparable to that of the EFFBS algorithm (see Table 2):

$$Time_{\{EFFBS\}} = 101\times10^{10}\,\mathrm{MUL} + 26\times10^{10}\mathrm{DIV} + 99\times10^{10}\mathrm{ADD}.$$

However, the amount of memory required by the EFFBS algorithm is 0.01 MB, which is negligible compared to the 20,000 MB (or 20 GB) required by the FFBS algorithm. The EFFBS algorithm represents a practical approach for designing low-footprint intrusion detection systems.

5. Conclusion

This paper considers the memory problem of the forward–backward (FB) algorithm when trained with a long observation sequence. This is an interesting problem facing various HMM applications ranging from computer security to robotic navigation systems and bioinformatics. To alleviate this issue, an Efficient Forward Filtering Backward Smoothing (EFFBS) algorithm is proposed to render the memory complexity independent of the sequence length, without incurring any considerable computational overhead and without any approximations. Given an HMM with N states and an observation sequence of length T, both FB and EFFBS algorithms have the same time complexity, $\mathcal{O}(N^2T)$. Nevertheless, FB has a memory complexity of $\mathcal{O}(NT)$, while EFFBS has a memory complexity that is independent of T, $\mathcal{O}(N)$. EFFBS requires fewer resources than FB, yet provides the same results.

Appendix A. FB & FFBS algorithms

Algorithm 2: Forward-Scaled($o_{1:T}, \lambda$)

Output: $\alpha_t(i) \triangleq P(q_t = i, o_{1:t} \mid \lambda)$, $\ell_T(\lambda)$ and $scale()$

for $i = 1, \ldots, N$ **do** /* Initialization */
  $\alpha_1(i) \leftarrow \pi_i b_i(o_1)$
$scale(1) \leftarrow \sum_{i=1}^{N} \alpha_1(i)$
for $i = 1, \ldots, N$ **do**
  $\alpha_1(i) \leftarrow \alpha_1(i)/scale(1)$
for $t = 1, \ldots T-1$ **do** /* Induction */
  for $j = 1, \ldots, N$ **do**
    $\alpha_{t+1}(j) \leftarrow \sum_{i=1}^{N} \alpha_t(i) a_{ij} b_j(o_{t+1})$
  $scale(t+1) \leftarrow \sum_{i=1}^{N} \alpha_{t+1}(i)$
  for $i = 1, \ldots, N$ **do**
    $\alpha_{t+1}(i) \leftarrow \alpha_{t+1}(i)/scale(t+1)$
for $t = 1, \ldots T$ **do** /* Evaluation */
  $\ell_T(\lambda) \leftarrow \ell_T(\lambda) + log(scale(t))$

Algorithm 3: Backward-Scaled ($o_{1:T}, \lambda, scale()$)

Output: $\beta_t(i) \triangleq P(o_{t+1:T} \mid q_t = i, \lambda)$

for $i = 1, \ldots, N$ **do** /* Initialization */
  $\beta_T(i) \leftarrow 1$
for $t = T-1, \ldots, 1$ **do** /* Induction */
  for $i = 1, \ldots, N$ **do**
    $\beta_t(i) \leftarrow \frac{\sum_{i=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{scale(t+1)}$

Algorithm 4: Forward-Filtering($o_{1:T}, \lambda$)

Output: $\gamma_{t|t}(i) = P(q_t = i \mid o_{1:t}, \lambda)$ and $\ell_T(\lambda)$

for $i = 1, \ldots, N$ **do** /* Initialization */
  $\gamma_{1|0}(i) = \pi_i$
for $t = 1, \ldots T-1$ **do** /* Induction */
  for $i = 1, \ldots, N$ **do**
    $\gamma_{t|t}(i) \leftarrow \frac{\gamma_{t|t-1}(i) b_i(o_t)}{\sum_{j=1}^{N} \gamma_{t|t-1}(j) b_j(o_t)}$
  for $j = 1, \ldots, N$ **do**
    $\gamma_{t+1|t}(j) \leftarrow \sum_{i=1}^{N} \gamma_{t|t}(i) a_{ij}$
for $t = 1, \ldots T$ **do** /* Evaluation */
  $\ell_T(\lambda) \leftarrow \ell_T(\lambda) + \log \sum_{i=1}^{N} b_i(o_t) \gamma_{t|t-1}(i)$

Algorithm 5: Backward-Smoothing $(\gamma_{T|T}, \lambda)$

Output: $\gamma_{t|T}(i) = P(q_t = i \mid o_{1:T})$ and

$$\xi_{t|T}(i,j) = P(q_t = i, q_{t+1} = j \mid o_{1:T})$$

1 $\gamma_{T|T} \leftarrow input$ /* Initialization */

2 **for** $t = T-1, \ldots, 1$ **do** /* Induction */

3 **for** $i = 1, \ldots, N$ **do**

4 **for** $j = 1, \ldots, N$ **do**

5 $\xi_{t|T}(i,j) \leftarrow \frac{\gamma_{t|t}(i)a_{ij}}{\sum_{i=1}^{N}\gamma_{t|t}(i)a_{ij}}\gamma_{t+1|T}(j)$

6 $\gamma_{t|T}(i) \leftarrow \sum_{j=1}^{N}\xi_{t|T}(i,j)$

References

Askar, M., Derin, H., 1981. A recursive algorithm for the bayes solution of the smoothing problem. IEEE Trans. Automatic Control 26 (2), 558–561.

Baum, L.E., 1972. An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. In: Proc. 3rd Symposium Inequalities III, Univ. Calif., Los Angeles, Calif., 1969, 1–8 (dedicated to the memory of Theodore S. Motzkin).

Baum, L.E., Petrie, G.S., Weiss, N., 1970. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. Annal. Math. Statist. 41 (1), 164–171.

Baum, L.E., Petrie, T., 1966. Statistical inference for probabilistic functions of finite state markov chains. Annal. Math. Statist. 37, 1554–1563.

Cappe, O., 2001. Ten years of HMMs. <http://www.tsi.enst.fr/cappe/docs/hmmbib.html>.

Cappe, O., Moulines, E., 2005. Recursive computation of the score and observed information matrix in hidden markov models. In: IEEE/SP 13th Workshop on Statistical Signal Processing, 2005, pp. 703–708.

Chang, R., Hancock, J., 1966. On receiver structures for channels having memory. IEEE Trans. Inform. Theory 12 (4), 463–468.

Churbanov, A., Winters-Hilt, S., 2008. Implementing em and viterbi algorithms for hidden markov model in linear memory. BMC Bioinform. 9 (224) (it has supplement file for formulas).

Coppersmith, D., Winograd, S., 1990. Matrix multiplication via arithmetic progressions. J. Symbolic Comput. 9, 251–280.

Dempster, A., Laird, N., Rubin, D., 1977. Maximum likelihood estimation from incomplete data via the EM algorithm. J. Roy. Statist. Soc., Ser. B 39 (1), 1–38.

Devijver, P., 1985. Baum's forward–backward algorithm revisited. Pattern Recognition Lett. 3, 369–373.

Elliott, R.J., Aggoun, L., Moore, J.B., 1995. Hidden Markov Models: Estimation and Control. Springer-Verlag.

Ephraim, Y., Merhav, N., 2002. Hidden markov processes. IEEE Trans. Inform. Theory 48 (6), 1518–1569.

Florez-Larrahondo, G., 2005. Incremental learning of discrete hidden Markov models. Ph.D. Thesis, Mississippi State University, Department of Computer Science and Engineering.

Grice, J., Hughey, R., Speck, D., 1997. Reduced space sequence alignment. CABIOS 13 (1), 45–53.

Koenig, S., Simmons, R.G., 1996. Unsupervised learning of probabilistic models for robot navigation. Proc. – IEEE Internat. Conf. on Robotics Automat. 3, 2301–2308.

Krogh, A., Brown, M., Mian, I.S., Sjolander, K., Haussler, D., 1994. Hidden Markov-models in computational biology applications to protein modeling. J. Mol. Biol. 235 (5), 1501–1531.

Lane, T.D., 2000. Machine learning techniques for the computer security domain of anomaly detection. Ph.D. thesis, Purdue University, W. Lafayette, IN.

LeGland, F., Mevel, L., 1997. Recursive estimation in hidden markov models. In: 36th IEEE Conf. Decision Control, San Diego, CA, December.

Levinson, S.E., Rabiner, L.R., Sondhi, M.M., 1983. An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition. Bell System Tech. J. 62, 1035–1074.

Li, Z., Evans, R., 1992. Optimal filtering, prediction and smoothing of hidden markov models. In: Proc. of the 31st IEEE Conf. on Decision and Control, vol. 4, 1992, pp. 3299–3304.

Lindgren, G., 1978. Markov regime models for mixed distributions and switching regressions. Scandinavian J. Statist. 5, 81–91.

Mann, T.P., 2006. Numerically stable hidden markov Model implementation, an HMM scaling tutorial, February.

Meyer, I.M., Durbin, R., 2004. Gene structure conservation aids similarity based gene prediction. Nucleic Acids Res. 32 (2), 776–783.

Miklos, I., Meyer, I.M., 2005. A linear memory algorithm for Baum–Welch training. BMC Bioinform. 6, 231.

Narciso, T.L., 1993. Adaptive channel/code matching, Ph.D. thesis. University of Southern California, California, United States.

Ott, G., 1967. Compact encoding of stationary markov sources. IEEE Trans. on Inform. Theory 13 (1), 82–86.

Rabiner, L., 1989. A tutorial on Hidden Markov Models and selected applications in speech recognition. Proc. IEEE 77 (2), 257–286.

Raviv, J., 1967. Decision making in markov chains applied to the problem of pattern recognition. IEEE Trans. Inform. Theory 13 (4), 536–551.

Shue, L., Anderson, D., Dey, S., 1998. Exponential stability of filters and smoothers for hidden markov models. IEEE Trans. Signal Process. 46 (8), 2180–2194.

Sivaprakasam, S., Shanmugan, S.K., 1995. A forward-only recursion based hmm for modeling burst errors in digital channels. In: IEEE Global Telecommunications Conference, vol. 2, GLOBECOM '95, 1995, pp. 1054–1058.

Tarnas, C., Hughey, R., 1998. Reduced space hidden markov model training. Bioinformatics 14, 401–406.

Wang, W., Guan, X.-H., Zhang, X.-L., 2004. Modeling program behaviors by hidden Markov models for intrusion detection. Proc. 2004 Internat. Conf. on Machine Learning Cybernet. 5, 2830–2835.

Warrender, C., Forrest, S., Pearlmutter, B., 1999. Detecting intrusions using system calls: alternative data models. In: Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, USA, pp. 133–145.

Yeung, D.-Y., Ding, Y., 2003. Host-based intrusion detection using dynamic and static behavioral models. Pattern Recognition 36 (1), 229–243.